# CSIC Project Guidelines

How to engineer your project

# Preliminaries

- Each team should have 3 members
  - 1 x team leader role
  - 2 x technical roles
- Team leader
  - Organizational responsibility
  - Work breakdown and assignment
  - Development work
- Technical members
  - Feedback to team leader
  - Development work

# Phases of the process

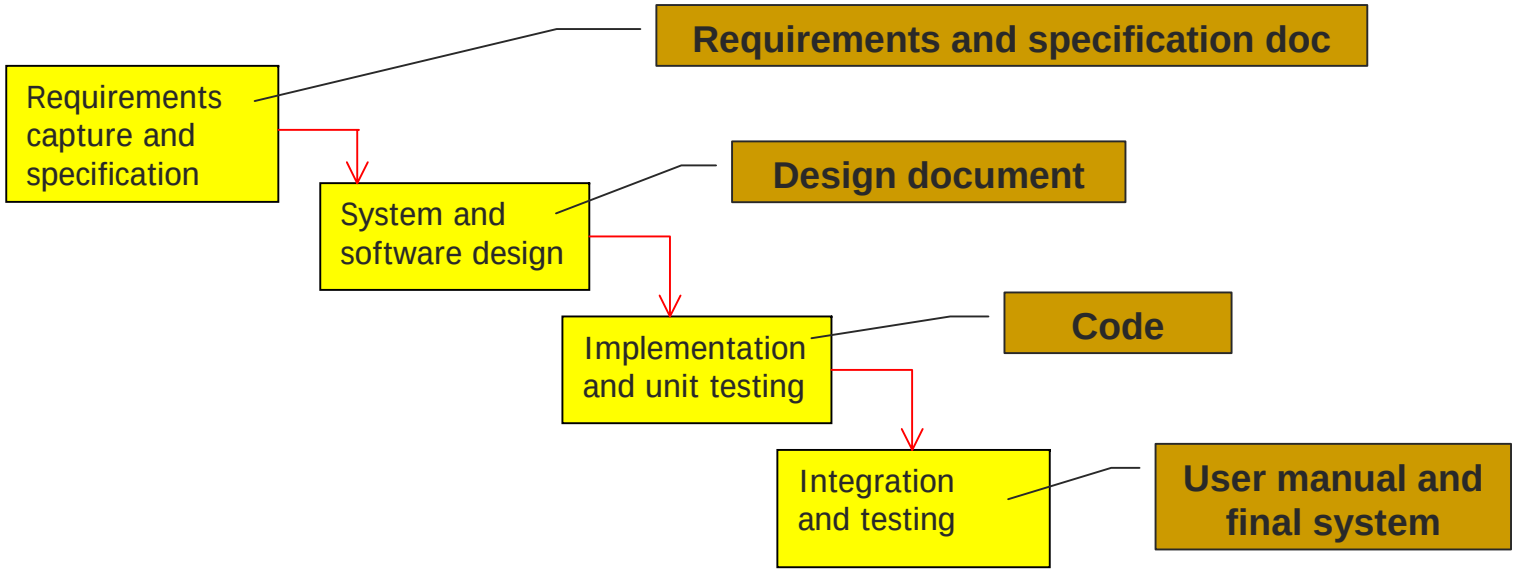
- Analysis
  - Requirements capture
  - Specification
- Design
  - Functional decomposition (if any)
  - UML Design
- Implementation
- Testing (validation and verification)

# Interaction in phases

1) Problem statement
   - Coordinator → Team
2) Requirements and specification
   - Team → Coordinator
3) Repeat step 2 to satisfaction of coordinator
4) Code
   - Internal for team
5) Testing
   - Internal for team
6) Project documentation and working system
   - Team → Coordinator

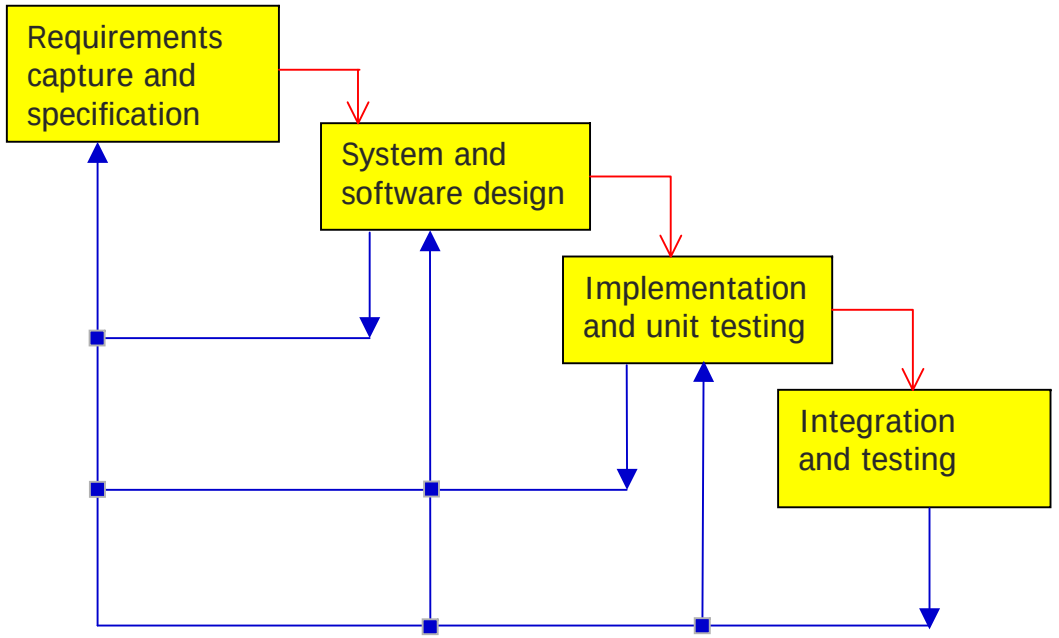
# The process itself

- Our projects are very simple
- So the waterfall model should suffice



# Another possibility

- In case you need to iterate
  - Document the source and target phases
  - Generate an “engineering change notice”



# Requirements & specifications

- List requirements in a numbered hierarchical format
- You may also use UML use-cases
- Express the specification using DFDs
- Give requirements and specifications in the same document
- Try to finalize your requirements
  - Requirements are like water, they're easier to build on when they're frozen

# Design

- You should ideally use UML
- Use class diagrams to generate the static code structure of your application
- Define your interfaces clearly
- Delineate the responsibility of various classes and components explicitly
- Document
  - Functional units (what everything is supposed to do)
  - Interfaces (what each function expects, any pre-conditions, any post-conditions, errors generated)
- Act in haste and repent at leisure, code too soon and debug forever

# Coding

- Code cleanly
  - Any fool can write code that a computer can understand, good programmers write code that humans can understand
- Pick a coding convention and stick to it
- Comment a lot
- Divide work up according to functional units
- Have a multi-file project, it will help
- Handle non-nominal cases (i.e.: errors)
- Use Eclipse (an advisory, in your own interest)
- Keep your UI functional and simple

# Integration and testing

- Should be easy if you documented your interfaces well
- Be careful of how/where your errors are propagated and handled
- Try to be as foolish as possible when testing your system
  - S/W engineering is a battle between programmers making their programs more and more fool-proof, and the universe, making bigger and bigger fools
  - So far, the universe is winning
  - Your project coordinators will be the fools when testing your system

# Delivery

- Expected on time
- Should include
  - Working system
  - User manual
  - A README on how to install and any extraneous requirements to run it
- Good luck!